

Why Safety Reporting IS like Rocket Science

David J. Garbutt, BSI AG, Baden-Dättwil, Switzerland

ABSTRACT

Safety Reporting (SR) doesn't have a reputation for being exciting except for those moments of terror when a mistake is discovered. I will explore this and other ways that safety programming and rocket science are similar and draw lessons that question how we validate our programming, to enjoy a successful launch without spending 28 Billion dollars.

INTRODUCTION

I am going to argue we¹ need a different approach to safety reporting. Why?

We have the same problems in SR we had 20 years ago when I first came into the pharmaceutical industry.

- ❑ "Validation" –What exactly does it guarantee? Is that actually what we want?
*"Why is this listing wrong if you say the program has been validated?"*²
- ❑ No regression testing.
"Everything was fine, now it doesn't work! Aha, you updated the macro library".
- ❑ Incomplete or ambiguous specifications.
"Ah, so, 'include pre-baseline values' means include pre-baseline values if they were recorded less than 4 weeks prior to first drug"

TYPICAL SOFTWARE DEVELOPMENT

A common model (typical of classic waterfall models) is:

1. Requirements
2. Design
3. Code
4. Build
5. Test (go back to 3 if fail)
6. β Release
7. Final release
8. Sales Push, Web Site.
9. Money comes in
10. Minimal maintenance given the level of competition.

There are now other more agile methods for managing programming projects, but these are even less relevant to the standard reporting example because they emphasize prototyping and incremental development.

SOFTWARE DEVELOPMENT FOR SAFETY REPORTING

See this very different scheme:

9. Specs made according to 'standards', extended or modified by whim
8. coding and checking by programmer
7. database changes, additional tables
6. modified headers, footers, subset, formatting
5. dry run and test (lots of eyeballing and cross checking with specifications)
4. Checks of code or by re-programming by independent programmer
3. data base lock
2. N³ days for validation.
1. Ignition - Lift OFF

Steps 7 to 4 are actually run in parallel in most cases. Step 1 (and step 8) may also be out of sequence in the worse cases.

WHAT IS DIFFERENT?

Write Once

¹ When I say we I mean 'we programmers working in this area in the Pharmaceutical industry'. I am not referring to any particular client I have worked at recently, or at any time. There may be places where some of these problems do not apply, but to my knowledge no-one has solved all of these issues.

² Twenty years ago the complaint might have been "when you said it was working", the content is unchanged.

³ N is a small integer, certainly less than 30 and tending, independently of workload, towards 0 at a rate proportional to M⁻², where M is the inertial mass of your management.

PhUSE 2006

Unlike normal software development the ideal for SR is 'write once, use once' and also unlike normal software the meaning of a validation can only be in relation to a given set of data. And there is only one set that really matters – the set we end up getting. But what exactly what this set is, is unknowable at the time we write our programs.

Nature or Nurture?

This is a token for an old scientific (and sociological and psychological) debate about the nature of humanity. Are we a clean slate when we are born?, or are we determined by the genes we get? The outcome of that debate is now clear⁴, the dichotomy is meaningless in the sense that the answer is both. We are given genes and what they can (and do) do is filtered by our environment. They are not independent routes of cause – but connected serially (to use an electrical analogy).

The debate about should we validate programs or listings is of the same type. The outputs we build are a product of *both* programs, data and their interaction and we cannot simplify the problem by only considering one part while holding the other part constant. This means our process needs to be very different from the development of normal software.

Conclusion

Safety reporting **is** like rocket science because all the testing and development *must* be done before launch, and after that we don't want to revisit any programming. In other words we cannot rely on a long alpha and beta release program to resolve quality issues. We have to test before we get the final versions of our data and we need to make tests as automated as possible and to test the product directly and minimize the (slow) validation processes on outputs generated after final data are released.

Consequences

The rest of this paper will lay out what I believe the consequences of this surprising insight are, and make suggestions for how we can develop further safety reporting with these redefined goals and insights in mind.

SOFTWARE DEVELOPMENT MODELS

SOFTWARE DEVELOPMENT AT NASA

The SEL has achieved productivity comparable to the average MIS or IT system, at the same time achieving quality levels that are at least 10 to 20 times better. To put it a little differently, the average MIS shop would need about 14 calendar months and 110 staff-months to deliver a 100,000 line-of-code MIS system, and it would typically contain about 850 defects when delivered. The NASA SEL would deliver a system of that size with about the same amount of time and effort, but it would contain only about 50 defects.

Source: <http://www.stevemccconnell.com/sqcrib.htm>

RELIABILITY OF A SYSTEM AND ITS COMPONENTS

Simple engineering theory about reliability is useful: here failure rates multiply so high quality depends on high quality components.

TESTING NOT VALIDATION

Validation is more of a legal term than a quality control process. I believe that to solve the problems of SR we need the approach of Rocket Science – and that means developing and using routinely testing tools independent of the programming process. One way to do this is to have double programming, another way is to develop tools to read our outputs and make checks. Perhaps there are other ways to tackle the problems and control the risks and we should certainly have that debate. But first I want to make some suggestions towards instituting more testing into our processes.

Test, Test, and Test again

- ❑ Write tests that work from the outputs directly as well as the data, have general checks of text that do not depend on reading the data or on the method of programming.
- ❑ Although it is true that data management are responsible for cleaning data: some of that task is conditioned on what programming we do and the assumptions our software makes. So it makes sense to add independent checks running off our products.
- ❑ People are very good at finding errors and if we add tools to help with manual checking
- ❑ Share code, share tests
- ❑ Add each new test to a common pool,
- ❑ Measure error and issue rates as well as deadlines.
- ❑ Use process control plots on error rates, publish them on the intranet
- ❑ Consider re-testing old deliverables, as calibration
- ❑ Always include integrity checks in your programming (eg never assume a key variable is unique and fix the issue by using `proc sort nodupkey.`)
- ❑ Always plot your data – develop QC plots for difficult data – eg labs. Run these repeatedly as data arrive.
- ❑ Include comparisons against previous versions of outputs – catch re-introduced bugs (regression testing). Information about exactly when errors appeared is not important for management – but vital for diagnosis
- ❑ If outputs have much formatting that changes (eg run date is on every page) then filter these lines out or create a simplified version of the listing purely for data comparison and then monitor changed lines in a separate comparison.
- ❑ Use metadata on populations included in each listing to automate checks of patient counts etc.
- ❑ Reward people that find errors. Do not solely praise staff that deliver on time!
- ❑ In short, integrate testing into production. As Rocket Scientists it is the only way!

⁴ See EO Wilson, Consilience, chapter 7.

PhUSE 2006

DOUBLE PROGRAMMING

Extreme programming has made a stir recently with its strange rules and apparent perverse bucking of all the trends but there are some things to listen to. One of the main pieces is double programming: that is programming is always done by two people sitting next to each other and working together. Some interesting claims are made about the effectiveness of this concept at reducing logical bugs – those errors of understanding that are the hardest to find.

CONCLUSIONS

I would christen this approach TOD (test-oriented development) if that acronym didn't spell death in German.

SUMMARY:

- ☐ Automated testing,
- ☐ Eliminate process steps requiring manual checking,
- ☐ Automated transformation of specs to programs.

CONTACT INFORMATION

I would value your comments and questions on this paper. Please contact the author at:

David J Garbutt
BSI AG
Täferstrasse 16A
CH – 5405 Baden-Dättwil
Work Phone: +41 56 484 1920
Fax: +41 56 484 1930
Email: d.garbutt@bsiag.com
Web: www.bsiag.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration.

Other brand and product names are trademarks of their respective companies.